RAWDATA Assignment 5 – Information Retrieval

The purpose of this assignment is to start practicing with IR as well as to indicate directions for treating IR-extensions in the Portfolio project. To start working with Portfolio relevant issues, download the file words.sql and import it in MySQL to your local portfolio database. You will also need the slides from the IR-presentation. The code from these can also be found in a separate SQL-file: *IRcode_from_slides.sql*.

Your hand-in is due on November 13 on Moodle. One submission per group.

Question 1

- IIR Exercises 1.1, 1.2 and 1.3

Question 2

- IIR Exercises 2.1, 2.2 , 2.3 , 2.8 and 2.9

Question 3

Consider the words-table. Apart from providing a combined inverted index for posts and comments, the words table also encodes additional info about field (what) and position of the word (sen, idx – respectively sentence and word number starting with respectively 0 and 1) as well as the following lexical info: word category / part-of-speech (pos) and lemma (lemma).

- What would be a candidate key for the words table and how can you verify that it actually is a candidate key by analysis of the content of the table?
- Notice that a reasonable assumption would be that a lemma should be the same for a given word and a specific category. How can this assumption be verified?
- Does it imply any dependencies that we can assume to hold? What or which?
- If you'd identified one or more functional dependencies in b), what would a normalization of the words relation lead to?

Question 4

In this exercise, we will build and use a positional inverted index on posts.

a) Use the table words as source to build a positional inverted index on the title column in posts. Store this index in a new table and call it **mwi** (my word index). You'll need idx but you can ignore the sen column as we assume that titles always consists of a single sentence. You can also ignore the lexical info from the words table. Consider only words purely formed by letters, that is, any character of the word should be among A-Z or a-z. HINT: the simplest approach here is to use the "create table xx as select …" construct in SQL.

Use your new post title positional inverted index to answer the following queries. Don't use any other tables to answer the queries.

b) What are the words starting with 'data' in the post titles?
c) What are the words appearing in titles together with the word 'database'?
d) What are the words appearing in titles after the word 'database'?
e) What are the words appearing in titles immediately after the word 'database'?
f) What are the words appearing in titles within a proximity of 3 words (no more than 3 words away) from the word 'database'?
g) Extract a concordance for the word 'database', that is, a list of contexts for the word, where context here means the word together with the immediately preceding and the immediately succeeding word.

Question 5

In this exercise, we will use **mwi** again and build another inverted index **mwib** on the body column of posts. The main challenge here is to consider our small new database consisting of these two tables and to query and optimize this to efficiently process certain queries.

a) Use again the table words as source to build a positional inverted index on the body column in posts. Store this index in a new table and call it **mwib** (my word index on body). You'll need the idx as well as the sen column since body-values may include multiple sentences. You can again ignore the lexical info and restrict to words purely formed by letters as in **mwi**.

Notice that you now have two tables without any indexing. You can use the "create index …." command to add indexing to a table ("alter table" if you want to introduce primary key, but this is not necessary here). "Drop index" can be useful when experimenting.

b) Consider and try to process the query:
select m1.id, m2.word, m2.idx from mwib m1, mwib m2
where m1.id=m2.id and m1.sen =m2.sen and m1.word ='database';
You probably get bored waiting for the answer. What can you do?

c) Consider and try to process the query:
select mwi.word from mwi
where mwi.word not in (select mwib.word from mwib where mwi.id=mwib.id);
Explain the answer to this. Do you need further optimization to process this? Why / Why not?

d) The following (rather weird) query retrieve id and word for posts where at least one word appears in the same position in the title and in some sentence in the body. (Useful? – maybe only in this context as an example).
select mwib.id, mwib.word from mwi, mwib
where mwi.id=mwib.id and mwi.idx=mwib.idx and mwi.word=mwib.word;
Try to process it. Stop the processing if you get impatient. What can you do to avoid waiting for the answer?

Question 6

The section "Preparing the data" in the IR-slides gives an example of data preparation.

a) Explain why the result conforms to the description on the slide and give your own description of what exactly is the result from executing this SQL-code.

b) The SQL-code leads to an inverted indexing where all posts, questions as well as answers, are individually indexed. Suppose you rather would prefer to index question posts only, but in such a way that every word from the question as well as the answers referring to this are entered in the index. Write preparation code in SQL to generate such an alternative inverted index.

Question 7

a) Assume the existence of a table "stopwords" listing stop-words. How could you use such a table and what would be the impact of using it?

b) Download the file stopwords.sql from Moodle and import it to your database to get a set of candidate stopwords. Modify the preparation code from the slides as well as the alternative code from 2b) so that you take stop-words into consideration.

Question 8

Consider the bestmatch3 procedure code in the slides section "Ranking and Ordering".

a) What are the possible values for the rank?

b) Explain why the SQL expression lead to the desired ranking.

c) Modify the code such that it takes 5 rather than 3 keywords.

Question 9

The section in the IR-slides "Dynamic procedure – towards a solution" includes code to a stored procedure called rewrite. The dummy replacement substrings can be substituted by partial SQL expressions.

a) Try to develop a stored procedure bestmatch that can **generate a string representing an SQL-expression** as in bestmatch3, but with any number of arguments (2 or more).
b) Consider the example code for a simple find-procedure in the IR-slides' section "Dynamic procedure – towards a solution". With inspiration from this, try to modify your code from a) so that the SQL-expression is executed when the procedure is called.